How do I do that in Arcpy:

Illustrating classic GIS Tasks

Arthur J. Lembo, Jr.

Arthur J. Lembo, Jr.
2017

First Printing: January, 2017

Arthur J. Lembo, Jr.
1440 East Sandy Acres Drive
Salisbury, MD 21804

www.artlembo.com

Preface

In 2004, I created a small publication with my students titled *How Do I Do that In ArcGIS/Manifold*. It was an enormously fun endeavor, which to my surprise really took off in the GIS community. There were tens of thousands of downloads, some lengthy debate, and finally some attempts to replicate the document for other GIS software products. Seeing the document take on a life of its own was really gratifying. This current document is a continuation of that theme, but with a focus on the use of Arcpy to accomplish the tasks.

Once again, we will revisit the 1988 United States Geological Survey (USGS) classic document titled *The Process for Selecting Geographic Information Systems*[1] (Guptil, et. al., 1988). As you might recall from the previous *How Do I Do That* documents, the USGS report provided an overview of the process for selecting geographic information systems, in addition to a checklist of functions that a GIS should include. The functions were broken into five separate categories: user interface, database management, database creation, data manipulation and analysis, and data display and presentation.

As the title indicates, I envision this book to act as a sort of reference to the question of *how do I do that..*, residing on the user's lap while attempting to implement GIS functionality with Arcpy and ArcGIS 10.3. Also, I believe simply perusing through the pages will be convincing enough, allowing users to consider the use of Arcpy as part of their daily GIS activities. As a teaching tool, one can see how many of the same Arcpy functions are just reused in a different fashion to complete a task, without the need to run some kind of special wizard or new tool. So, if you can write Python code, you can build all kinds of functionality in GIS.

You will also notice that this book is much shorter than the original *How do I do that in ArcGIS/Manifold*, and without illustrations. This was done on purpose to keep the cost of production low and to allow users to quickly get an answer to their Arcpy questions. I believe that users will continue to make use of this book to accomplish the tasks using Arcpy. Finally, all of the examples were tested using ArcGIS 10.3. A .zip file of the file geodatabase is downloadable from my blog: artlembo.com. Therefore, users can recreate the Arcpy commands from the book using the actual data. To do that of course, the code in this book must be retyped. This too was done on purpose - retyping the commands are the best way to learn. Simply copying and pasting the code is not going to help you learn

[1] Guptill, S., D. Cotter, R. Gibson, R. Liston, H. Tom, T. Trainer, H. VanWyhe. 1988. "A Process for Selecting Geographic Information Systems". Technology Working Group – Technical Report 1. USGS Open File Report 88-105.

what is actually happening. So, while there is a little more effort on the part of the reader, I believe it will be the most effective way to learn how to write Arcpy scripts. As you become more accustomed to scripting with Arcpy, you will find that you begin to *think in Arcpy*. For me, when presented with a GIS conundrum, I constantly find myself thinking about the Arcpy solution, rather than the classical GIS commands.

Python is a very easy to understand scripting language, even for those who are unfamiliar with programming. When I began this book, I had no idea whether Arcpy could complete a majority of the tasks using only the Python API engine. I was quite pleased to see that virtually all of the tasks listed in *How do I do that in ArcGIS/Manifold* could actually be accomplished with Arcpy. In fact, of all the *How do I* books, Arcpy was able to accomplish the most tasks as defined in the USGS document.

Finally, I would be remiss if I did not acknowledge the excellent support community from Esri, gis.stackexchange.com, and many friends, as they provided great advice and direction when I often found myself stuck.

I welcome you to participate in the discussion of this book on my blog: **artlembo.wordpress.com**.

Arthur J. Lembo, Jr.
January, 2017

Table of Contents

How to understand this guide vi
Adding a column to a table 2
Sorting tabular or graphical data 3
Calculating values for new fields using arithmetic or related tables - making field calculations. 4
Relating data files and fields 5
Database Creation 6
Digitizing 7
Assigning Topology - identifying intersection points 9
Creating a Polygon from Line Segments 9
Creating a distance buffer from line segments 10
Correcting topological errors - eliminating overlaps, undershoots, and dangles. 10
Import and export - importing database tables, raster data, and vector data 11
Data Manipulation and Analysis 12
Vector Retrieval - Select by Window 13
Raster Retrieval - select data by area masks 13
Data restructuring - convert from raster to vector; and vector to raster 14
Modify raster cell size by resampling 15
Reducing unnecessary coordinates - weeding 15
Smoothing data to recover sinuosity 15
Data restructuring - changing raster values by selected area or feature 16
Generate a TIN from point data 16
Kriging from point data (not illustrated) 16
Generate contour data from points (not illustrated) 17
Generate contour data from raster (not illustrated) 17
Data Transformation - mathematical transformation of raster data (not illustrated) ..... **Error! Bookmark not defined.**
Projection definition and coordinate transformation 18

Vector overlay - polygon in polygon overlay; point in polygon overlay; line in polygon overlay 18
Raster processing - mathematical operations on one raster (sine, cosine, exponent) 19
Raster processing - mathematical operations on two rasters - adding, subtracting, minimum, maximum 19
Neighborhood functions - average, minimum, maximum, most frequent 20
Statistical functions - calculating areas, perimeters and lengths 21
Cross tabulation of two data categories 21
General - specify distance buffers; polygons within a distance of selected buffers; find nearest features 22
3D analysis - generating slope and aspect 22
Identifying watersheds 23
Network functions - choosing the optimal path through a network 23
Defining a drive-time zone 23
Geocoding addresses (not illustrated) 23
Topological intersection 24
Intersect 24
Union 24
Identity 24

How to understand this guide

This guide follows the topic headings from the book *How do I do that in ArcGIS/Manifold*, as a way to illustrate the capabilities of Arcpy in ArcGIS for accomplishing classic GIS tasks. You will notice that some gaps exist where the tasks cannot be completed using Arcpy, and would require the GUI. In some instances the gaps are logical as the specified task requires user interaction. In other cases, however, the gap exists because a function that one might assume would be a logical addition to the software simply was not built into the Arcpy interface.

All of the Arcpy scripts were tested in ArcGIS 10.3, on a geodatabase that you can download from artlembo.wordpress.com. All of the code was written in the most straightforward way possible – however, as you learn Python and Arcpy, you will discover other ways to accomplishing the tasks. In many cases, there may be more efficient ways to accomplish the task.

I have attempted to format the code as best as possible so that it is easier to read. However, there are some examples that have too long of names, or too much code to fit on a single line on the page. In these cases, there may be some minor *wrapping* of the text.

Finally, in most instances, I have added the syntax of the Arcpy function in a lighter gray so that readers can refer to the full syntax when wanting to add more optional statements.

Database Management

Database management functions provide for tracking, retrieval, storage, update, protection, and archiving of stored data.

page 29. The Process for Selecting Geographic Information Systems

Adding a column to a table

Adding a column to a table is relatively straightforward in Arcpy. The original *How do I do that in ArcGIS/Manifold* document did not include options like changing the name, data type, or size of the columns. Nonetheless, the following shows a few examples on how to modify tables using the `AddField_management` function in Arcpy.

To add a new column (in this case, a column named **homeage** that stores Integers, the user simply writes:

Add a column

```
AddField_management(in_table, field_name, field_type, {field_precision}, {field_scale}, {field_length},
{field_alias}, {field_is_nullable}, {field_is_required}, {field_domain})
```

```
arcpy.AddField_management("parcels","homeage","LONG")
```

Rename a column

```
AlterField_management(in_table, field, {new_field_name}, {new_field_alias}, {field_type}, {field_length},
{field_is_nullable}, {clear_field_alias})
```

```
arcpy.AlterField_management("parcels","homeage","ageofhome")
```

Alter the type

ArcGIS does not allow altering the field type if the field is already populated. Therefore, I show a two-step process of adding a new field and calculating the values of the original field in to the new field. Notice that the clause to calculate the field requires an exclamation point (!) around the field name.

```
arcpy.AddField_management("parcels","newhomeage","TEXT")

arcpy.CalculateField_management("parcels","newhomeage","!homeage!","PYTHON_9.3")
```

Remove a column

In this example, multiple fields are removed, and placed in a Python list surrounded by brackets ([…])

```
arcpy.DeleteField_management("parcels",["newhomeage","homeage"])
```

Sorting tabular or graphical data

An Arcpy SearchCursor function will return the values from a table. In this case, you can specify the fields you want returned and also indicate the fields you want to sort the data by. In this example, we are sorting the records by the land (in ascending order) and the assessment value (in descending order).

Sort in descending order

```
arcpy.SearchCursor(dataset, where_clause=None, spatial_reference=None, fields=None, sort_fields=None)

SearchCursor(dataset, {where_clause}, {spatial_reference}, {fields}, {sort_fields})

mysc = arcpy.SearchCursor("parcels","","","","acres D")
```

Sort multiple fields

```
mysc = arcpy.SearchCursor("parcels","","","","land A; asmt D")
```

Sort using an on-the-fly calculation

ArcGIS does not have the capability to perform an on-the-fly calculation.

Calculating values for new fields using arithmetic or related tables - making field calculations.

New values may be calculated using the `CalculateField_management` statement.

```
CalculateField_management(in_table, field, expression, {expression_type}, {code_block})
```

The following example updates a field named homevalue in our parcels table by subracting the value of the land from the entire assessment value of the property:

```
arcpy.CalculateField_management("parcels","homeage","!ASMT! - !LAND!","PYTHON_9.3")
```

Calculations on a related table

Arcpy does not have a direct way to perform a calculation on a related field.

Calculations without updating the table

Arcpy does not have a direct way to perform a calculation on the fly without updating the table.

Relating data files and fields

This example illustrates how to relate multiple tables together based on a related item. While many complex relates are possible, simple tables are used here to illustrate the process. For example, we can join two tables (parcels and propclas) as:

```
AddJoin_management(in_layer_or_view, in_field, join_table, join_field, {join_type})
```

```
arcpy.AddJoin_management("parcels","parcelkey","parvalues","parcelkey")
```

Special relationships between tables in the ArcGIS GUI under "Add Relations" are not currently supported with Arcpy in ArcGIS 10.3 The `AddRelate_management` function is available in ArcGIS Pro.

Database Creation

Database creation functions are those functions required to convert spatial data into a digital form that can be used by a GIS. This includes digitizing features found on printed maps or aerial photographs and transformation of existing digital data into the internal format of a given GIS.

Page 29, The Process for Selecting Geographic Information Systems

Digitizing

Ordinarily, digitizing is performed within the GUI of a GIS where the user points-and-clicks on the screen. However, Arcpy can be used if you have a table of coordinate values you want to enter, or if you are receiving input from say an Internet based application. The following examples illustrate how to insert geometries into an existing layer.

Points are created from two coordinate values by making a point with Arcpy, establishing a cursor for the shape field, and then inserting the row as:

```
Point({X}, {Y}, {Z}, {M}, {ID})
```

```
mypt = arcpy.Point(5997611.48964, 2069897.7022)
mycursor = arcpy.da.InsertCursor("trees",["SHAPE@"])
mycursor.insertRow([mypt])
```

Or, if you had latitude and longitude values (SRID 4326), and wanted to insert them into the elevpts layer (EPSG: 2261) table, you could issue the following:

```
mypt = arcpy.Point(-76.477,42.447)
myptg = arcpy.PointGeometry(mypt,4326)
pt_spc = myptg.projectAs(arcpy.SpatialReference(2261))
mycursor = arcpy.da.InsertCursor("trees",["SHAPE@"])
mycursor.insertRow([pt_spc])
```

Adding Lines

Lines are created from a series of point geometries. To add a Line, you could string together a series of Point geometries. In the following example we are a line to the roads layer:

```
mypoints = [[834818, 890000],[880100, 890100],[890200, 890600]]
for pt in mypoints:
        coordarray.append(arcpy.Point(pt[0],pt[1]))
myline = arcpy.Polyline(arcpy.Array(coordarray))
mycursor = arcpy.da.InsertCursor("roads",["SHAPE@"])
mycursor.insertRow([myline])
```

Adding Polygons

Polygons are constructed in a similar manner, but with the Polygon object, where the first and last coordinate are identical:

```
mypoints =[[834818,890000],[880100,890100],[890200,890600], [834818, 890000]]
for pt in mypoints:
          coordarray.append(arcpy.Point(pt[0],pt[1]))
mypol = arcpy.Polygon(arcpy.Array(coordarray))
mycursor = arcpy.da.InsertCursor("parcels",["SHAPE@"])
mycursor.insertRow([mypol])
```

Assigning Topology - identifying intersection points

```
Intersect_analysis(in_features;in_features..., out_feature_class, {join_attributes}, {cluster_tolerance}, {output_type})
```

You can find intersection points for either line or area geometries in a single layer as:

```
arcpy.Intersect_analysis(["roads","roads"],"rdint","ONLY_FID",0.1,"POINT")
```

Intersections on multiple layers

Intersection points for multiple layers are found by:

```
arcpy.Intersect_analysis(["flood","firm"],"ptint","ONLY_FID",.1,"POINT")
```

Creating a Polygon from Line Segments

```
FeatureToPolygon_management(in_features;in_features..., out_feature_class, {cluster_tolerance}, {attributes}, {label_features})
```

```
arcpy.FeatureToPolygon_management(["roads"],"roadpoly")
```

Creating a distance buffer from line segments

```
Buffer_analysis(in_features, out_feature_class, buffer_distance_or_field, {line_side}, {line_end_type},
{dissolve_option}, {dissolve_field;dissolve_field...}, {method})
```

Buffers can be created on any type of geometry, either points, lines, or areas - using the Buffer_analysis statement. In this case, you can either enter a distance value or a field value:

```
arcpy.Buffer_analysis("flood","floodbuf",1000)
```

Creating ringed buffers around a geometry:

```
MultipleRingBuffer_analysis(Input_Features, Output_Feature_class, Distances;Distances..., {Buffer_Unit},
{Field_Name}, {Dissolve_Option}, {Outside_Polygons_Only})
```

```
arcpy.MultipleRingBuffer_analysis("flood","floodring",[200,400,600,800])
```

Correcting topological errors - eliminating overlaps, undershoots, and dangles.

```
TrimLine_edit(in_features, {dangle_length}, {delete_shorts})

ExtendLine_edit(in_features, {length}, {extend_to})
```

```
arcpy.TrimLine_edit("roads",10,"DELETE_SHORT")

arcpy.ExtendLine_edit("roads", "10 Feet", "EXTENSION")
```

Import and export - importing database tables, raster data, and vector data

Database Tables

```
TableToTable_conversion(in_rows, out_path, out_name, {where_clause}, {field_mapping}, {config_keyword})
```

In this example we import a .dbf file into a feature table. However, you can import other files such as csv, txt, etc.:

```
arcpy.TableToTable_conversion("c:/temp/goodpts.dbf","c:/temp/","newpts")
```

Export Vector Data

```
FeatureClassToFeatureClass_conversion(in_features, out_path, out_name, {where_clause}, {field_mapping},
{config_keyword})
```

In this example, we export two feature classes (roads and parks) to shapefiles:

```
arcpy.FeatureClassToShapefile_conversion(["roads","parks"],"c:/temp/output/")
```

Data Manipulation and Analysis

Data manipulation and analysis functions provide the capability to selectively retrieve, transform, restructure, and analyze data.

Retrieval options provide the ability to retrieve either graphic features or feature attributes in a variety of ways. Transformation includes both coordinate/projection transformations and coordinate adjustments. Data restructuring includes the ability to convert vector data to raster data, merge data, compress data, reclassify or rescale data, and contour, triangulate, or grid random or uniformly spaced z-value data sets

Analysis functions differ somewhat depending on whether the internal data structure is raster or vector based. Analysis functions provide the capability to create new maps and related descriptive statistics by reclassifying and combining existing data categories in a variety of ways. Analysis functions also support: replacement of cell values with neighboring cell characteristics (neighborhood analysis); defining distance buffers around points, lines and areas (proximity analysis); optimum path or route selection (network analysis); and generating slope, aspect and profile maps (terrain analysis).

Page 29, The Process for Selecting Geographic Information Systems.

Vector Retrieval - Select by Window

Although *select by window* normally assumes an interactive session with the GUI, you can use Arcpy to select by geometric shape. In this example, we are entering polygon exressed as a rectangle. All vector features intersecting the box are selected using the `SelectLayerByLocation` which offers many selection options like CONTAINS, TOUCHES, etc.:

```
SelectLayerByLocation_management(in_layer, {overlap_type}, {select_features}, {search_distance},
{selection_type}, {invert_spatial_relationship})
```

```
mypoints = [[834818, 890000],[880100, 890100],[890200, 890600], [834818, 890000]]
for pt in mypoints:
     coordarray.append(arcpy.Point(pt[0],pt[1]))
mypol = arcpy.Polygon(arcpy.Array(coordarray))
mysel = arcpy.SelectLayerByLocation_management("trees","COMPLETELY_WITHIN",mypol)
```

Raster Retrieval - select data by area masks

We can extract raster data using other layers, geometries, or even circles and rectangles. The following command shows how to extract a raster based on a layer:

```
ExtractByMask(in_raster, in_mask_data)
```

```
myras = arcpy.sa.ExtractByMask("u27elu","parks")
```

We can also create a geometry value (polygon), and pass that value into the ExtractByMask command to extract the raster (notice this is similar to the creation of the polygon earlier):

```
coordarray = []
mypoints = [[380000, 4697300],[381000, 4697300],[381000, 4699300],[380000, 4697300]]
for pt in mypoints:
     coordarray.append(arcpy.Point(pt[0],pt[1]))
mypol = arcpy.Polygon(arcpy.Array(coordarray))
myras2 = arcpy.sa.ExtractByMask("u27elu",mypol)
```

Data restructuring - convert from raster to vector; and vector to raster

Raster to Vector

Floating point data cannot be converted to vector objects. So, in the following example, we will reclassify the DEM into three different categories. Then, we can convert the raster to polygons:

```
Reclassify(in_raster, reclass_field, remap, {missing_values})

newras = arcpy.sa.Reclassify("u27elu","Value",arcpy.sa.RemapRange([[0,500,1],[500,1000,2],[1000,1500,3]]))

arcpy.RasterToPolygon_conversion("newras","raspoly","SIMPLIFY","Value")
```

Vector to Raster

```
PolygonToRaster_conversion(in_features, value_field, out_rasterdataset, {cell_assignment},
{priority_field}, {cellsize})

parkras = arcpy.PolygonToRaster_conversion("parks","size","parkraster")
```

Modify raster cell size by resampling

There are a number of ways to modify a raster cell (BILINEAR, CUBIC, NEAREST NEIGHBOR, etc.). Those ways are found in the `resampling_type`. The following example rescales the DEM from 10m grid cells to 50m grid cells:

```
Resample_management(in_raster, out_raster, {cell_size}, {resampling_type})
```

```
arcpy.Resample_management("u27elu","u27_50m",50,"BILINEAR")
```

Reducing unnecessary coordinates - weeding

```
Generalize_edit(in_features, {tolerance})
```

You can simplify a geometry using the Douglas-Peucker algorithm as:

```
arcpy.Generalize_edit("roads",10)
```

Smoothing data to recover sinuosity

```
SmoothLine_cartography(in_features, out_feature_class, algorithm, tolerance, {endpoint_option},
{error_option})
```

```
arcpy.SmoothLine_cartography("roads","road_smooth","BEZIER_INTERPOLATION",10)
```

Data restructuring - changing raster values by selected area or feature

```
PolygonToRaster_conversion(in_features, value_field, out_rasterdataset, {cell_assignment},
{priority_field}, {cellsize})
```

Raster data can be altered where they intersect features in another raster layer as:

```
parkras = arcpy.PolygonToRaster_conversion("parks","park_no","parkrast")

outraster = arcpy.sa.Con(arcpy.sa.Raster("parkrast") > 2, 122.0,"u27elu")
```

Generate a TIN from point data

```
arcpy.CreateTin_3d(out_tin=None, spatial_reference=None, in_features=None, constrained_delaunay=None)
CreateTin(out_tin, {spatial_reference}, {in_features;in_features...}, {constrained_delaunay})
```

```
arcpy.CreateTin_3d("mytin","","elevpts height")
```

Kriging from point data (not illustrated)

```
arcpy.Kriging_3d(in_point_features=None, z_field=None, out_surface_raster=None, semiVariogram_props=None,
cell_size=None, search_radius=None, out_variance_prediction_raster=None)
Kriging(in_point_features, z_field, out_surface_raster, semiVariogram_props, {cell_size}, {search_radius},
{out_variance_prediction_raster})
```

Kriging has a rather sophisticated interface that is worth learning. However, basic kriging can be performed as followings, allowing ArcGIS to optimize many of the kriging parameters:

```
arcpy.Kriging_3d("elevpts","Height","krigsurf","SPHERICAL",30)
```

Generate contour data from points (not illustrated)

ArcGIS does not generate contour data directly from points. To complete this task, you must create a raster interpolation (i.e. kriging) from the points, and then create contour data from the raster (see below).

Generate contour data from raster (not illustrated)

```
arcpy.Contour_3d(in_raster=None, out_polyline_features=None, contour_interval=None, base_contour=None,
z_factor=None)
Contour(in_raster, out_polyline_features, contour_interval, {base_contour}, {z_factor})
```

```
arcpy.Contour_3d("krigsurface","contour20",20)
```

Projection definition and coordinate transformation

Layers may be projected and defined on-the-fly with Arcpy. The following query projects the parks layer, stored as *State Plane, NY Central*, to the *WGS 84* coordinate system (4326).

Change Projection

```
Project_management(in_dataset, out_dataset, out_coor_system, {transform_method;transform_method...},
{in_coor_system}, {preserve_shape}, {max_deviation})
```

```
arcpy.Project_management("parks","parksll",arcpy.SpatialReference(4326))
```

Define Projection

```
DefineProjection_management(in_dataset, coor_system)
SpatialReference({item})
```

If the geometry is in the correct numerical format, but does not have a coordinate system assigned, you can assign a coordinate system. For example, assume that the parcel layer does not have a coordinate system defined, but should actually be UTM 18N:

```
arcpy.DefineProjection_management("parksll",arcpy.SpatialReference(3450))
```

Vector overlay - polygon in polygon overlay; point in polygon overlay; line in polygon overlay

```
SelectLayerByLocation_management(in_layer, {overlap_type}, {select_features}, {search_distance},
{selection_type}, {invert_spatial_relationship})
```

There are numerous overlap types that Arcpy will allow a user to perform when overlaying and finding geometry features contained within polygons. The SelectLayerByLocation function allows for selecting among points, lines, and polygons.

```
arcpy.SelectLayerByLocation_management("trees","COMPLETELY_WITHIN","parks")
```

in this case, it does not matter whether the trees are points, lines, or polygons.

Raster processing - mathematical operations on one raster (sine, cosine, exponent)

Arcpy implements numerous mathematical operations on raster data using the Spatial Analyst (sa) module:

Sine

```
new_surface_sin = arcpy.sa.Sin("u27elu")
```

Cosine

```
new_surface_cos = arcpy.sa.Cos("u27elu")
```

Exponent

```
new_surface_pow = arcpy.sa.Power("u27elu",2)
```

Raster processing - mathematical operations on two rasters - adding, subtracting, minimum, maximum

Arcpy has extensive map algebra functions that can work on multiple rasters. The key is to identify the correct function to perform the task. The most straightforward way is to use the arithmetic symbols that you are already familiar with on raster defined feature:

Addition

```
newras = arcpy.Raster("u27elu") + arcpy.Raster("parkras")
```

Subtraction

```
newras = arcpy.sa.Minus("u27elu","parkras")
```

Multiplicationn

```
newras = arcpy.Raster("u27elu") * arcpy.Raster("parkras")
```

Neighborhood functions - average, minimum, maximum, most frequent

The FocalStatistics function in the Spatial Analyst module allows for the calculations of neighborhood functions like minimum, maximum, sum, mean, etc.

```
FocalStatistics(in_raster, {neighborhood}, {statistics_type}, {ignore_nodata})
```

Minimum

```
raster_mean = arcpy.sa.FocalStatistics("u27elu","","MINIMUM")
```

Sum

```
raster_mean = arcpy.sa.FocalStatistics("u27elu","","SUM")
```

Maximum

```
raster_mean = arcpy.sa.FocalStatistics("u27elu","","MAXIMUM")
```

Mean

```
raster_mean = arcpy.sa.FocalStatistics("u27elu","","MEAN")
```

Statistical functions - calculating areas, perimeters and lengths

Descriptive statistics on geometries can be calculated to include area, length, or perimeter. In the example below, we add a new field to the Parks layer and calculate the area in hectares:

```
arcpy.AddField_management("parks","parkarea","FLOAT")
arcpy.CalculateField_management("parks","parkarea","!SHAPE.AREA@HECTARES!","PYTHON_9.3")
```

Cross tabulation of two data categories

Cross tabulation matrices are created with two functions. In the following example, we will first create the tabulation of the number of trees, by condition within each park. Then, we will take the resulting tabulation and convert the data into a pivot table:

```
TabulateIntersection_analysis(in_zone_features, zone_fields;zone_fields..., in_class_features, out_table,
{class_fields;class_fields...}, {sum_fields;sum_fields...}, {xy_tolerance}, {out_units})

PivotTable_management(in_table, fields;fields..., pivot_field, value_field, out_table)
```

```
arcpy.TabulateIntersection_analysis("parks","name","trees","park_tree","cond")

arcpy.PivotTable_management("park_tree","name","cond","PNT_COUNT")
```

General - specify distance buffers; polygons within a distance of selected buffers; find nearest features

The ability to specify distance buffers was already addressed in a previous section. However, finding polygons within a specified distance and finding nearest features are calculated as:

```
SelectLayerByLocation_management(in_layer, {overlap_type}, {select_features}, {search_distance},
{selection_type}, {invert_spatial_relationship})
```

Features within a distance

```
arcpy.SelectLayerByLocation_management("trees","WITHIN_A_DISTANCE","parks",1000)
```

Nearest Features

```
Near_analysis(in_features, near_features;near_features..., {search_radius}, {location}, {angle}, {method})
```

```
arcpy.Near_analysis("elevpts","trees",10000,"NO_LOCATION","NO_ANGLE","PLANAR")
```

3D analysis - generating slope and aspect

Slope

```
newslope = arcpy.sa.Slope("u27elu")
```

Aspect

```
newaspect = arcpy.sa.Aspect("u27elu")
```

Identifying watersheds

```
Watershed(in_flow_direction_raster, in_pour_point_data, {pour_point_field})
```

```
mywatershed = arcpy.sa.Watershed(arcpy.sa.FlowDirection("u27elu"),"elevpts","ID")
```

Network functions - choosing the optimal path through a network

Creating a network feature dataset in ArcGIS is beyond the scope of this document. However, using the Paris.gdb from the ESRI tutorial, the following Arcpy commands will allow one to build the network of streets, add stores to the streets, and then route over the entire network. Note that the network dataset ParsiMultimodal_ND already exists:

```
arcpy.na.MakeRouteLayer("ParisMultimodal_ND","parisnetwork","Meters","FIND_BEST_ORDER","PRESERVE_BOTH")
arcpy.na.AddLocations("parisnetwork","","Stores")
arcpy.na.Solve("parisnetwork")
```

Defining a drive-time zone

Similar to the previous example, this example will use the Paris.gdb which already includes a network dataset. In this example, we define 1, 2, and 3 minute drive time zones from each store:

```
arcpy.na.GenerateServiceAreas("Stores","1 2 3", "Minutes","ParisMultimodal_ND","newservice")
```

Geocoding addresses (not illustrated)

Geocoding addresses require a geocoding dataset, or an ArcGIS Online account that keeps track of tokens, and is not illustrated in this document.

Topological intersection

Intersect

```
Intersect_analysis(in_features;in_features..., out_feature_class, {join_attributes}, {cluster_tolerance}, {output_type})
```

The basic Intersection analysis is called as:

```
arcpy.Intersect_analysis(["firm","parcels"],"firmout","ALL")
```

However, using variables in Arcpy, we can do more sophisticated topological intersections like intersecting all the parcels with only flood areas (the firm layer) that are "AE". First, we select those "firm" features with a ZONE = 'AE', and then performing the topological intersection:

```
firmAE = arcpy.MakeFeatureLayer_management("firm","firmAE","ZONE = 'AE'")

arcpy.Intersect_analysis([firmAE,"parcels"],"parcelsint","ALL")
```

Union

```
arcpy.Union_analysis([firmAE,"parcels"],"parcelsunion","ALL")
```

Identity

```
arcpy.Identity_analysis(firmAE,"parcels","firmidentify","ALL")
```